1987-03

The application of single-source shortest path algorithms to an OJSC contingency planning model and a vehicle routing model

Brown, Jerome W. Jr.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE APPLICATION OF SINGLE-SOURCE SHORTEST PATH ALGORITHMS TO AN OJCS CONTINGENCY PLANNING MODEL AND A VEHICLE ROUTING MODEL

by

Jerome W. Brown, Jr.

March 1987

Thesis Advisor: R. E. Rosenthal

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable) 55	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) THE APPLICATION OF SINGLE-SOURCE SHORTEST PATH ALGORITHMS TO AN OJCS CONTINGENCY PLANNING MODEL AND A VEHICLE ROUTING MODEL.

12 PERSONAL AUTHOR(S) BROWN, Jerome W., Jr.

13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day) 1987 March	15 PAGE COUNT 52

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Shortest Path, Algorithms, reference node aggregation, SOTACA model, Vehicle Routing Model

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis investigates the use of single-source shortest path algorithms in two unrelated contexts. In the first application, the label setting and label correcting algorithms are examined for applicability to and implementation within a J-8, Organization of the Joint Chiefs of Staff contingency planning model. This model has encountered problems of slow execution directly related to shortest path computations, which can be resolved by the methods proposed. Additionally, these two shortest path algorithms are examined for use within the model for identification and presentation of alternate optima when they exist.

The second application involves the development of a new algorithm, called reference node aggregation, which is designed to efficiently produce a subset of the all-pairs shortest path solution for large scale networks. The anticipated use of this algorithm is in connection with

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL R. E. Rosenthal	22b TELEPHONE (Include Area Code) 408-646-2795	22c OFFICE SYMBOL 55R1

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

19. Abstract (cont).

Vehicle Routing Model. The motivation for producing a subset of the full solution is that only a very small subset of all possible pairs of nodes will ever be considered for consecutive visitation by a vehicle; hence, most of the information in an all-pairs solution is irrelevant. For those pairs whose exact shortest paths are not computed, a single-step approximation is devised which does not require access to peripheral storage. The new algorithm has three user-specified engineering parameters whch effectively contol the tradeoff between the accuracy of the subset solution and the effort required to compute it.

The Application of Single-Source Shortest Path Algorithms to an OJCS Contingency Planning Model and a Vehicle Routing Model

by

Jerome W. Brown Jr.
Captain, United States Marine Corps
B.S., Miami University, 1980
M.S., University of Southern California, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1987

ABSTRACT

This thesis investigates the use of single-source shortest path algorithms in two unrelated contexts. In the first application, the label setting and label correcting algorithms are examined for applicability to and implementation within a J-8, Organization of the Joint Chiefs of Staff contingency planning model. This model has encountered problems of slow execution directly related to shortest path computations, which can be resolved by the methods proposed. Additionally, these two shortest path algorithms are examined for use within the model for identification and presentation of alternate optima when they exist.

The second application involves the development of a new algorithm, called reference node aggregation, which is designed to efficiently produce a subset of the all-pairs shortest path solution for large scale networks. The anticipated use of this algorithm is in connection with vehicle routing models. The motivation for producing a subset of the full solution is that only a very small subset of all possible pairs of nodes will ever be considered for consecutive visitation by a vehicle; hence, most of the information in an all-pairs solution is irrelevant. For those pairs whose exact shortest paths are not computed, a single-step approximation is devised which does not require access to peripheral storage. The new algorithm has three user-specified engineering parameters which effectively control the tradeoff between the accuracy of the subset solution and the effort required to compute it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. THESIS CONTENT AND ORGANIZATION

This thesis investigates the use of single-source shortest path algorithms in two separate contexts. The first application, found in Chapter II, is embedded within a J-8, Organization of the Joint Chiefs of Staff (OJCS) planning program called State of the Art Contingency Analysis (SOTACA). SOTACA is an interactive, automated tool that assists staff planners and operations officers in quickly analyzing alternate plans for a contingency operation [Ref. 1: page 1-2]. SOTACA frequently computes shortest paths which are used in the routing of friendly forces over a network that represents the area of operation. The shortest path software currently contained in SOTACA is deficient in two respects:

(1) it is too slow, and

(2) it ignores alternate shortest paths when they exist.

Chapter II describes an implementation of single-source shortest path algorithms which resolves these two problems.

The second half of this thesis considers shortest path calculations embedded within vehicle routing problems over large networks. Whenever two nodes are visited consecutively by the same vehicle, the shortest path connecting those nodes must be used. However, only a minute fraction of all possible pairs of nodes will ever be considered for consecutive visitation. Therefore, it is desirable to avoid computing all possible shortest paths. For this reason, Chapter III develops an effective way of providing the shortest path information needed for vehicle routing without solving a large number of shortest path problems.

B. NETWORK FLOW MODEL STRUCTURE AND NOTATION

For the purpose of the discussions to follow, the network will be considered to be a directed graph $G = (N,A)$ consisting of a set of nodes, N,

$$N = \{ 1,2,3, \ldots n \}$$

and a set of arcs (ordered pairs of nodes),

$$A \subset N \times N,$$

where x denotes the Cartesian product. Nodes represent places (or items) of interest to the modeler while an arc defines the existence of a valid route (or relationship) between nodes. Associated with each arc is a nonnegative flow parameter, C(i,j), which is the cost assessed per unit of flow across the arc (i,j).

II. GLOBAL CONTINGENCY PLANNING AND NETWORKS

A. CONTINGENCY PLANNING WITH NETWORKS

The Organization of the Joint Chiefs of Staff (OJCS) uses network flow models in the conduct of global contingency planning. This is accomplished with the assistance of a modern-aid-to-planning-program (MAPP) called the State of the Art Contingency Analysis (SOTACA) model. One of SOTACA's principal functions provides for the representation of the area of operation in its various dimensions (i.e., land, sea, economic, political, . . .) enabling the commander and his staff to systematically analyze the mission and situation of the assigned task [Ref. 1: page 2-7]. This representation of the operational area takes the form of a network flow model where nodes represent places of significance and arcs between nodes represent movement paths for forces.

The primary function of SOTACA's network flow model is to enable the study and analysis of candidate routes (i.e., shortest paths) for movement of enemy and friendly forces on an administrative or tactical march. Associated with each arc in the network are two separate flow costs:

- the physical length in kilometers of the arc
- the time in minutes to traverse the arc

These two flow costs enable staff planners to study both time and distance in contingency planning.

B. PROBLEM DEFINITION

1. Background

In 1985, SOTACA was delivered to the J-8 OJCS by Science Applications International Corporation and shortly thereafter, follow-on documentation was initiated. This included an analyst's guide to using SOTACA with emphasis on the the network flow model and shortest path computations. Concurrent to the follow-on documentation, J-8 planners were trained to use SOTACA and started to analyze contingency plans. The subsequent use of SOTACA resulted in problems where model execution time increased alarmingly when the network approached the maximum allowable size of 300 nodes and 1250 arcs [Ref. 1]. (This is considered small by contemporary standards [Ref. 2,3].)

During May-June 1986, the author was assigned to the J-8 OJCS as part of his Naval Postgraduate School Operations Analysis experience tour. He was tasked by J-8 with determining:

(1) the shortest path methodology implemented in SOTACA, and

(2) if this methodology was responsible for the increased execution times.

The author's analysis determined that SOTACA uses an implementation of Floyd's all-pairs shortest path algorithm [Ref. 4: page 210]. Subsequent discussion between J-8 analysts and the author brought to light that only a small portion of the all-pairs solution is ever used by SOTACA. The results of further analysis concluded that the slow execution of the SOTACA model is rooted in both the network data structure supporting the implementation of Floyd's algorithm and in the overabundance of information it produces.

A secondary issue which surfaced during this analysis was a perceived shortcoming of SOTACA's shortest path implementation, namely, the nonrecognition of alternate shortest paths when they exist. J-8 analysts felt that the identification and use of alternate paths could enhance the analysis of a contingency plan via SOTACA.

2. The Issues

The remainder of this chapter addresses two specific issues which were still unresolved at the conclusion of the author's experience tour. The first is to determine what can be done to reduce or eliminate SOTACA's slow execution time. The second is to identify a methodology for generating alternate shortest paths for use by SOTACA.

C. SOTACA IMPLEMENTATION OF FLOYD'S ALGORITHM

This section provides a brief description of the shortest path methodology and the associated data structure currently used by SOTACA.

1. Floyd's Algorithm

Floyd's algorithm produces an all-pairs shortest path solution by examining every path between two nodes and recording that path which has smallest total flow cost. This process is repeated for every pair of nodes in the network. Figure 2.1 outlines the SOTACA implementation of Floyd's algorithm.

Underlying SOTACA's use of Floyd's algorithm is a data structure which contains a description of the network and provides for recording of the computed shortest paths.

```
Input:
 (1) A variable, NODES, which specifies the total number of nodes in the
 network.

 (2) The network description in the form of related vectors FROM-NODE,
 TO-NODE, and FLOW-COST.

Output:
 (1) The PATH function.

 (2) The PATH-COST function.

1. Initialization
 a. PATH(i,j) = -1 , for all i and j ∈ N
 b. PATH-COST(i,j) = ∞ , for all i and j ∈ N
 c. For l = 1, |A| Do:
       (1) PATH(FROM-NODE(l),TO-NODE(l)) = 0
       (2) PATH-COST(FROM-NODE(l),TO-NODE(l)) = FLOW-COST(l)
    End do
 d. UPDATE-FLAG = ON

2. Enumeration
 while UPDATE-FLAG = ON, do
    UPDATE-FLAG = OFF
    For i = 1 to NODES, do
       For j = 1 to NODES, do
         For k = 1 to NODES, do
            If PATH-COST(i,k) + PATH-COST(k,j) < PATH-COST(i,j) then
                  PATH-COST(i,j) = PATH-COST(i,k) + PATH-COST(k,j)
                  PATH(i,j) = k
                  UPDATE-FLAG = ON
            Endif
         End do
       End do
    End do
 End while
```

Figure 2.1 SOTACA's Implementation of Floyd's Algorithm.

2. **Input Arrays (Network Representation)**

The network description is contained in four arc-length (denoted |A|) arrays. The entries in these arrays are related by position and define an arc in the network. These arrays identify the origin of an arc (FROM-NODE), the destination of an arc (TO-NODE), the distance flow cost (DISTANCE), and the traversal time flow cost (TIME). Outside of these positional relations, the arc information is in random order in the arrays. That is, the first arc input by the user is placed in the first position of the arrays, the second arc in the second position and so on.

3. **Output Arrays (Shortest Path Representation)**

The shortest path solutions are contained in two pairs (i.e., four total) of n by n matrices (where n = |N|). Each solution pair, while utilizing the same network, is for a separate flow problem. The first is concerned with the physical distance between nodes, while the second involves traversal-time between nodes.

Each solution pair utilizes the same method for storing shortest path solutions. Thus, a solution consists of a PATH-COST function which identifies the shortest path distance (i.e., the total flow cost) between any two nodes in the network, and a PATH function [Ref. 4: page 211] which specifies the sequence of nodes on each shortest path.

The PATH-COST function is an all-pairs version of the well-known label function [Ref. 2,5: page 15]. The PATH function is also well-known and Figure 2.2 depicts the iterative process for recovering shortest paths from it.

PATH(i,j) =	-1	No path exists from node i to node j
	0	Node j is connected to node i by arc (i,j).
	k	To reach node j from node i, first goto node k and then examine PATH(k,j) to determine which node is next on the path.

Figure 2.2 SOTACA PATH Function.

Armed with an understanding of how SOTACA currently produces and records shortest path information, attention is now turned to methods by which the network module deficiencies can be eliminated.

D. SINGLE-SOURCE SHORTEST PATH ALGORITHMS

SOTACA computes all-pairs shortest paths in a preprocessing method which typically requires a twenty minute waiting time (on a MICRO VAX) for networks at or near the maximum allowable size. As has been stated previously, SOTACA is able to perform its force routing function if it has at a minimum, the shortest path solution from a specified node to all other nodes in the network. In the literature available today, this type of problem is often referred to as a single-source shortest path problem [Ref. 4: page 203]. It is proposed that SOTACA use a single-source shortest path algorithm to compute shortest paths on-demand (i.e., as needed by the user). To this end, this section presents an examination of two well known single-source shortest path algorithms. These two algorithms are label setting and label correcting, and there are numerous variations of each.

Prior to presenting the algorithms, two changes to the SOTACA data structure are introduced. These modifications support the functioning of the single-source shortest path algorithms as well as provide a means to more efficiently represent network data and shortest path solutions in SOTACA.

1. Modifying SOTACA'S Network Data Structure

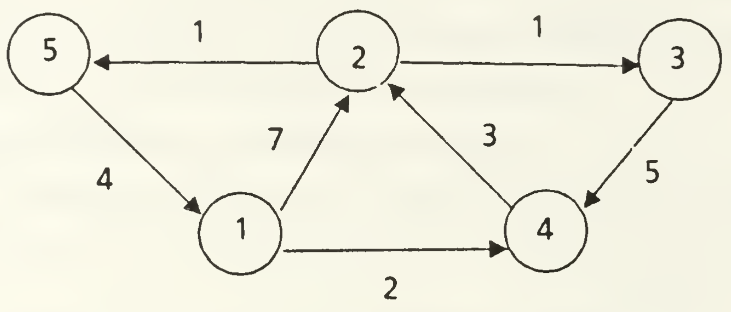
Two changes to SOTACA's data structure are proposed. The first concerns the manner in which the network description is represented internally, while the second involves the method for storing the shortest path solutions.

a. Reorganizing the Network Data

The network representation SOTACA uses is rather cumbersome when it comes to locating specific arc information. In locating the set of all arcs which originate from node i (this set is called the forward star of i [Ref. 3: page 218]), SOTACA conducts an arc-length search of the FROM-NODE array. This inefficiency is part of the larger problem that has surfaced with symptoms of slow execution.

This inefficiency can be overcome in two steps. First of all, the network data in the original arrays are put through a one-time sort which places the data in ascending order based upon the FROM-NODE field. As a result, the forward star of any node is in contiguous space within the arrays. Sequencing the arcs of the network by forward star yields efficiencies in solving for shortest paths.

At the completion of the one-time sort, the FROM-NODE array is used to construct an array known as the TAIL array [Ref. 2,5: page 13] which then replaces the FROM-NODE array. The TAIL array is of length $|N|+1$. TAIL(i) specifies the initial position of the contiguous space containing all arc information for arcs originating at node i, while TAIL(i+1)-1 specifies the last position. Figure 2.3 shows an example of network data being transformed from the original network representation to the forward star format.



FROM-NODE	1	2	3	1	4	5	2
TO-NODE	2	5	4	4	2	1	3
FLOW-COST	7	1	5	2	3	4	1

(a) SOTACA Network Representation

FROM-NODE	1	1	2	2	3	4	5
TO-NODE	2	4	5	3	4	2	1
FLOW-COST	7	2	1	1	5	3	4

(b) After the one-time sort

TAIL	1	3	5	6	7	8	
TO-NODE	2	4	5	3	4	2	1
FLOW-COST	7	2	1	1	5	3	4

(c) Forward Star Network Representation

Figure 2.3 Transformation of Network Data to Forward Star Format.

There are two advantages in using this modified data structure. The first is the memory savings that occurs when the arc-length vector FROM-NODE is replaced by the node-length vector TAIL (as most) networks are such that $|N| << |A|$). However, the second advantage far outweighs any other, as the accessing of specific arc information has been transformed from an arc-length search to an examination of only those arcs of present interest.

b. The Predecessor and Label Functions

The reader will recall that SOTACA stores an all-pairs shortest path solution in the two n by n matrices, PATH and PATH-COST. Solutions to the single-source shortest path problems can be compactly recorded in two 1 by n arrays called the predecessor function [Ref. 2: page 10] and the label function. The predecessor function, P(), is associated with a single source node (i.e., the root node) and contains a tree of shortest paths. The predecessor function differs from PATH in that it specifies the backpath from a node to the root node with each entry indicating which node was visited (coming from the root node) immediately prior to the current node. PATH, on the other hand, specifies a forward sequence of nodes for traversing the shortest path from one node to another. Figure 2.4 depicts the predecessor function.

The label function, U(), contains the total flow cost to reach a node from the specified root. Thus, it is the one row of the original all-pairs PATH-COST function associated with the root node r.

P(i) =	j	Node j immediately precedes node i on the backapth to the root node.
	0	Indicates that i is the root node if U(i) = 0. Otherwise indicates that i cannot be reached from the root node.

Figure 2.4 The Predecessor Function.

2. Label Setting Algorithms

One method of solving the single-source shortest path problem is to use a label setting algorithm [Ref. 2,3,4,5]. In this method, also known as Dijkstra's algorithm [Ref. 4: page 204], the nodes are partitioned into two sets, labeled and unlabeled. Labeled nodes are those for which the shortest path from the source is known, and unlabeled nodes are those for which it is not known. At each iteration, the method identifies the cheapest unlabeled node which can be reached from a labeled node, and adds this node to the labeled set. It should be noted that label setting (in contrast to label correcting) requires:

$$C(i,j) \geq 0 \text{ for all } (i,j) \in A$$

which was stated as an assumption in Chapter I.

The label setting algorithm, as with other single-source shortest path algorithms, generates a tree T consisting of a single root node r with other nodes connected to that root by some shortest path. Associated with each node i in the tree is a label that specifies the cost of the shortest path originating at the root node and ending at node i.

In the first iteration, the root node r is the only labeled node and it has a label of zero. At each iteration, the algorithm examines all the arcs originating at labeled nodes and identifies the unlabeled node (and the associated arc) which is cheapest to reach next. That unlabeled node is labeled and the associated arc added to the set of shortest paths, A_T. The algorithm repeats this process $|N|-1$ times since at each iteration one node is labeled. Figure 2.5 provides a step by step description of the label setting algorithm.

3. Improving the Label Setting Algorithm

It does not take much familiarization with the label setting algorithm to see that there is at least one major inefficiency with it. At each iteration but the first, the algorithm examines many arcs which it has examined previously, and some which have no bearing on producing new shortest paths (e.g., those arcs between labeled nodes). Thus, in a network of $|A|$ arcs, the algorithm ends up examining more than $|A|$ arcs. To avoid this extra work, the algorithm can be modified so that each arc is examined only once. [Ref. 5]

This is accomplished by setting temporary labels, which are intermediate guesses at the final (permanent) values of the labels. The algorithm proceeds similar to the basic algorithm. It starts with a specified root node and examines its forward star, setting all labels which can be improved upon and designating these labels as temporary. There are numerous ways to store temporary labels. The method chosen in this thesis was to use the sign bit of the predecessor function [Ref. 5]. Thus during any iteration, the sign of the predecessor function indicates the following:

$P(i) = 0$	indicates the label for node i has not been set
$P(i) < 0$	indicates the label for node i is temporary
$P(i) > 0$	indicates the label for node i is permanent

After the forward star has been examined and all temporary labels set, that node with the minimum temporary label is identified. This node's label is set permanent, and its forward star is examined thus repeating the process. The algorithm stops when all temporary labels have been set permanent.

```
Input:
  (1) A directed graph $G=(N,A)$ where $C(i,j) \geq 0$ for all i and $j \in N$.
  (2) A specified root node r.
Output:
  (1) The shortest path costs in the label function U().
  (2) The shortest path tree in the predecessor function P().

1. Initialize a tree $T(N_T,A_T)$ such that:
  a. $N_T = \{ r \}$
  b. $A_T = \{ \}$
  c. $U(t) = \infty$ for all $t \in N-N_T$
  d. $U(r) = 0$
  e. $P(t) = 0$ for all $t \in N$

2. Examine the forward star of all permanently labelled nodes and define:
  $S = \{(i,j): i \in N_T; j \in N-N_T, (i,j) \in A\}$
  IF $S = \{ \}$ THEN proceed directly to step 4.

3. To determine the "best" next label and its associated node, examine each element of S and:
  a. Find $(k,l) = \text{argmin} \{ U(i) + C(i,j) : (i,j) \in S \}$
  b. Redefine:
        $N_t = N_t \cup \{ l \}$
        $A_T = A_T \cup \{ (k,l) \}$
  c. Set:
        $P(l) = k$
        $U(l) = U(k) + C(k,l)$
  d. Repeat step 2.

4. Stop.

                                                        [Ref. 3,5]
```

Figure 2.5 The Label Setting Algorithm.

The basic label setting algorithm, presented in Figure 2.5, is easily modified to utilize this improvement. To do this, steps 2 and 3 of the algorithm are replaced by those shown in Figure 2.6.

To alter the basic algorithm, replace steps 2 and 3 in entirety by the following steps:

```
2. Examine the forward star of node r:
   IF $U(j) > U(r) + C(r,j)$
   THEN set:
       $U(j) = U(r) + C(r,j)$
       $P(j) = -r$
   ENDIF

3. IF $P(i) > 0$ for all $i \in N$
   THEN proceed to step 4
   ELSE set:
       $r = \text{argmin} \{ U(i) : P(i) < 0, \text{ for } i \in N \}$
       $P(r) = -P(r)$
       Repeat step 2
   ENDIF
```

Figure 2.6 Label Setting Improved by Use of Temporary Labels.

Label setting is just one single-source shortest path methodology. Attention is now turned to a related yet different single-source shortest path algorithm.

4. Label Correcting Algorithms

Another method of solving single-source shortest path problems is to use a label correcting algorithm [Ref. 3,4,5,6]. As with the label setting algorithm, label correcting generates a tree of nodes connected by shortest paths, and associates a label with each node in the tree. This node label is identical to that used in label setting.

Although not relevant in the present applications, it is worth noting that the label correcting algorithm can handle negative arc flow costs. There is one restriction, however: no cycle in the network can have negative total cost.

Label correcting sets temporary labels as it proceeds and upon reaching a specific ending condition declares all labels as permanent. To accomplish this, label correcting uses a list which contains nodes whose labels have been modified (i.e., corrected). Initially, this list contains only the root node r.

The list is processed in a last-in first-out (LIFO) fashion. When node i is stripped off of the list, its forward star is examined. If there exists a node j such that:

$$U(j) > U(i) + C(i,j) ,$$

then the label of node j is corrected by,

$$U(j) = U(i) + C(i,j) ,$$

and its predecessor function is updated,

$$P(j) = i.$$

In addition, a node whose label has been corrected is added to the list if not already appearing on it. The list is processed until there are no nodes left on it. At that time the algorithm stops and all labels are declared permanent. Figure 2.7 depicts the label correcting algorithm. [Ref. 3,5]

5. Improving the Basic Label Correcting Algorithm

There are numerous ways to improve on the basic label correcting algorithm. Most improvements specify a different manner in which to process the list of corrected nodes. One such method, termed scan-eligible [Ref. 6: page 67], uses a partitioning of the corrected nodes into two lists, NOW and NEXT. The nodes on NOW are processed in a LIFO fashion exactly as had been the basic algorithm's list. However, corrected nodes are placed onto the NEXT list. Then when all the nodes on NOW have been processed, NOW is set equal to NEXT, and NEXT is set to an empty set. This process is repeated until both NOW and NEXT are empty sets. The resulting labels are permanent at that time and the predecessor function contains the shortest path tree. Figure 2.8 depicts the label correcting algorithm improved through the use of scan-eligible lists.

E. MULTIPLE SHORTEST PATHS

The final SOTACA-related problem to be addressed concerns the existence of multiple shortest paths in a network. When multiple shortest paths exist, it is desired that the following occur:

- that a shortest path solution be produced, and
- that the multiple paths between a user-specified sink node and the root node be enumerated at the user's request. Note that in some networks the number of alternate shortest paths between a root and sink node may be very large. In

```
Input:
    (1) A directed graph $G=(N,A)$ with unbounded arc flow costs $C(i,j)$.
    (2) A specified root node r.
Output:
    (1) The shortest path costs in the label function $U()$.
    (2) The shortest path tree in the predecessor function $P()$.
1. Initialize a tree $T(N_T,A_T)$ such that:
    a. $N_T = \{ r \}$
    b. $A_T = \{ \}$
    c. $U(t) = \infty$ for all $t \in N-N_T$
    d. $U(r) = 0$
    e. $P(t) = 0$ for all $t \in N$
    f. LIST = { r }
2. IF LIST = { } THEN proceed directly to step 4
    ELSE define:
        i = node at the top of LIST
        LIST = LIST - { i }
    ENDIF
3. Examine the forward star of node i and for each node $j \in N$ where:
            $U(i) + C(i,j) < U(j)$
    Redefine:
            $N_T = N_T \cup \{ j \}$
            $A_T = \{ A_T - \{ (s,j) \in A_T \} \} \cup \{ (i,j) \}$
            $P(j) = i$
            $U(j) = U(i) + C(i,j)$
            LIST = LIST ∪ { j }
    Repeat step 2.
4. Stop.

                                                        [Ref. 3,5]
```

Figure 2.7   The Label Correcting Algorithm.

```
Input:
    (1) A directed graph $G=(N,A)$ with unbounded arc flow costs $C(i,j)$.
    (2) A specified root node $r$.

Output:
    (1) The shortest path costs in the label function.
    (2) The shortest paths in the predecessor function.

1. Initialize a tree $T(N_T, A_T)$ such that:
    a. $N_T = \{ r \}$
    b. $A_T = \{ \}$
    c. $U(t) = \infty$ for all $t \in N - N_T$
    d. $U(r) = 0$
    e. $P(t) = 0$ for all $t \in N$
    f. $NOW = \{ r \}$  $NEXT = \{ \}$
2. Define:
        $i =$ node at top of $NOW$
        $NOW = NOW - \{ i \}$
3. Examine the forward star of i and for each $j \in N$ where:
        $U(i) + C(i,j) < U(j)$
    Redefine:
            $N_T = N_T \cup \{ j \}$
            $A_T = \{ A_T - \{ (s,j) \in A_T \} \} \cup \{ (i,j) \}$
            $P(j) = i$
            $U(j) = U(i) + C(i,j)$
            $NEXT = NEXT \cup \{ j \}$
4. IF $NOW \neq \{ \}$ THEN repeat step 2.
    IF $NOW = \{ \}$ and $NEXT = \{ \}$ THEN goto step 5.
    Otherwise set:
            $NOW = NEXT$
            $NEXT = \{ \}$
    Repeat step 2
5. Stop.
                                                        [Ref. 3,5,6]
```

Figure 2.8 Improved Label Correcting Using Scan-Eligible Lists.

this case, it may be desirable to enumerate only a subset (size specified by the user) of all these alternate paths.

None of the algorithms discussed thus far are designed to produce a solution of this type. As designed, each algorithm merely provides the first shortest path solution encountered, ignoring any alternate shortest paths. This section describes some methods for attaining recognition of multiple paths and enumerating the alternate paths upon user request.

1. Determining the Existence of Multiple Shortest Paths

The first thing to be done is to find a means by which it can be determined that multiple shortest paths exist in a network. This is readily accomplished in the basic label setting algorithm through the setting of a flag to indicate that multiple shortest paths exist. This flag is set during the examination of arcs in the forward star of a labeled node. There are three conditions which indicate that multiple shortest paths exist and these are shown in Figure 2.9 . Note that these conditions can be looked for as the algorithm builds the shortest path tree.

```
When examining the forward star of node i:
    IF:
        $P(j) > 0$,
        $P(j) \neq i$, and
        $U(j) = U(i) + C(i,j)$
    THEN
        Set the Multiple Solution Flag
    ENDIF
```

Figure 2.9 Setting the Multiple Optimal Solution Flag.

The setting of a flag can also be done in the other three algorithms. However, the nature of improved label setting and both label correcting algorithms require an entire re-examination of all the arcs in the network after the shortest path tree has been built. If the conditions described in Figure 2.9 exist, then the multiple solution flag is set.

2. Enumerating the Multiple Shortest Paths

There are (at least) two methods by which multiple shortest paths from the root to a specified sink node can be identified and presented to the user: depth-first search [Ref. 7: page 91], and breadth-first search [Ref. 7: page 95]. For this thesis, the depth-first search technique is used.

The depth-first search is an optimistic approach which considers one path as good as any other during the search. After the designation of the root and sink node, a depth-first search starts at the root node and builds a tree as it searches to reach the sink node. Essentially, it adds nodes in a sequential fashion building a tree consisting of a main trunk with no branches. Thus, the search proceeds deeper and deeper (i.e., further away from the root node) until no more nodes can be added or when there is no hope of reaching the sink. At that point, the search backs up the tree one level and examines other alternatives that have not been searched at that level. If alternatives exist, the search goes back to its headlong dash down the new trunk. If no alternatives exist, the search will backup another level and check for alternatives there. The search stops when the sink node is found or when the backing up reaches the root node with no alternatives left unsearched.

Note that having a shortest path solution before enumerating the alternate shortest paths provides the known optimal distance to reach the sink, s. Thus, the depth-first search can be interrupted and directed to other paths once the current trunk length exceeds U(s) as there is no hope of reaching s optimally at that point. Figure 2.10 provides a description of the depth-first search algorithm.

F. TESTING AND EVALUATION

All six algorithms described in Section E (i.e., label setting, improved label setting, label correcting, improved label correcting, modified for multiple solutions label setting, and depth-first search) were successfully implemented in FORTRAN for execution on the Naval Postgraduate School's IBM-3033.

Testing and evaluation consisted of three specific stages. The first test was a minor one which simply verified the functioning and output of the single-source shortest path algorithms. The second test provided for a comparison of execution times for each of these algorithms against sample networks. The third test involved verifying the function and output of the modified for multiple solutions label setting and the depth-first search algorithms.

```
Input:

    (1) The root node r.
    (2) The sink node s.
    (3) MAXDEPTH = U(s).

Output:

    (1) The shortest paths from r to s, or an indication that r and s are not
connected via alternate paths.

1. LIST = { r }

2. IF LIST = { } THEN go directly to step 5.

3. Processing LIST in a LIFO fashion, define:

    a. i = node at the top of LIST

    b. IF U(i) > MAXDEPTH THEN:

           Remove i from LIST
           Repeat step 2

    c. IF i = s THEN

           Announce success
           Proceed directly to step 4

    d. Scan the forward star of node i for a successor node v:

           IF there are no eligible successors THEN

               Remove i from LIST
               Repeat step 2

           ELSE

               Add v to LIST
               Designate the associated arc as examined
               Set i = v
               Repeat step 2

           ENDIF

4. IF success has been announced THEN

           Record the path from r to s
           Remove s from the top of LIST
           Repeat step 2

    ELSE

           Proceed to step 5

    ENDIF

5. Stop.

                                                              [Ref. 7]
```

Figure 2.10 Depth-first Search Algorithm.

The sample networks used to test the algorithms were generated by the random network generator NETGEN [Ref. 8] on the IBM-3033. Each network consisted of a set of nodes and a set of directed arcs with their associated arc flow costs.

1. Label Setting and Label Correcting

a. Test for Algorithm Functioning

This test was designed to determine if the algorithms were properly implemented by verifying the contents of the predecessor and label functions. To accomplish this, NETGEN was used to generate small networks of 10 nodes and 30 arcs. Each algorithm was run using these networks and the resulting predecessor and label functions output for root nodes 1 through 10. The results were compared to the known results to verify the output accuracy. This test demonstrated that the algorithms functioned properly, and produced accurate predecessor and label functions.

b. Test for Algorithm Comparison

This test was designed to provide data in the form of algorithm execution times for a set of sample network problems. The times were compared to give an indication of the relative speed of each algorithm.

The test was designed as follows:

- The algorithms were standardized so that execution times measured the same set of tasks in each algorithm.
- NETGEN was used to generate 3 networks of the following sizes:

  Network 1: 300 nodes and 1250 arcs
  Network 2: 200 nodes and 1125 arcs
  Network 3: 100 nodes and 1000 arcs

- Each algorithm was run using the same set of 10 root nodes (one run per root per algorithm) with the execution times recorded internally (using FORTRAN GETIME and SETIME functions). The root nodes were generated using the FORTRAN pseudo-random number generator LRND.

The results of the test were summarized by noting the minimum, maximum, and mean execution times. These results are presented in Table 1 and provide a general indication of the speed of each algorithm.

The algorithms use a variety of variables and arrays to support the production of shortest paths. However, the bulk of the data structure in each algorithm is dedicated two functions:

- storing network data
- storing shortest path solutions

Disregarding the non-array variables, the approximate size of the data structure size for each algorithm, as well as that for Floyd's algorithm, is presented in Table 2 .

TABLE 1

EXECUTION TIMES FOR LABEL SETTING AND CORRECTING (IN CPU SECONDS)

	Problem Size		
Algorithm	100 nodes 1000 arcs	200 nodes 1125 arcs	300 nodes 1250 arcs
Label Setting			
Average	0.153	0.403	0.671
Minimum	0.133	0.372	0.139
Maximum	0.169	0.429	0.758
Improved Label Setting			
Average	0.010	0.066	0.150
Minimum	0.003	0.059	0.143
Maximum	0.026	0.083	0.183
Label Correcting			
Average	0.023	0.093	0.169
Minimum	0.009	0.059	0.139
Maximum	0.046	0.113	0.226
Improved Label Correcting			
Average	0.004	0.003	0.005
Minimum	0.003	0.003	0.003
Maximum	0.006	0.003	0.009

2. Depth-First Search For Multiple Paths

This test was primarily concerned with insuring that the multiple solution flag was set properly in the modified label setting algorithm and that the depth-first search algorithm produced the correct alternate paths for a given root and sink node.

The test was designed as follows:

- 2 networks of 10 nodes and 20 arcs were constructed. Network 1 was prepared such that only one shortest path existed between each pair of nodes. Conversely, network 2 was designed to have multiple shortest paths between some nodes. The shortest path solutions for both networks were known.

TABLE 2

ALGORITHM DATA STRUCTURE SIZE

Function	Label Setting	Label Correcting*	Floyd
Storing Network Data	2|A| + |N|	2|A| + |N|	3|A|
Shortest Path Generation	--	|N|	--
Shortest Path Solutions	2|N|	2|N|	2|N||N|
Total	2|A| + 3|N|	2|A| + 4|N|	3|A| + 2|N||N|

* The scan-eligible label correcting algorithm uses an additional (i.e., two total) |N|-length list for shortest path generation.

- The modified label setting algorithm was run with each node in the network specified as the root node for one run. The depth-first search algorithm was called after a shortest path solution had been produced if the multiple solution flag was set.
- The output of the depth-first search algorithm was compared to the known alternate shortest paths in the network. This comparison focused on accuracy of each alternate path produced as well as the completeness of the solution (with regards to the known quantity of alternate paths to be found).

Both the modification for multiple solutions and the depth-first search algorithm functioned properly. The multiple solution flag was set appropriately, and the depth-first search algorithm then produced the correct alternate paths.

III. REFERENCE NODE AGGREGATION

A. INTRODUCTION

The second part of this thesis involves questions which arose in connection with work by Professor Rosenthal on a vehicle routing algorithm. While not directly related to the SOTACA problem, the foundations are much the same. One main difference is that the networks of interest are large scale.

Despite this vast increase in problem size over SOTACA networks, the item(s) of interest is the same, namely the shortest paths between nodes.

B. PROBLEM DEFINITION

For consistency purposes, the same basic network terminology used in the SOTACA chapter will be used to formally describe the problem at hand.

1. Assumptions and Given Data

Let $G=(N,A,R)$ be a large-scale directed graph of $|N|$ nodes and $|A|$ arcs. The size of G is not fixed, but in practice it is expected that $|N|$ is large (e.g., 50,000 or more nodes), while $|A|$ is approximately bounded as follows:

$$1.75|N| < |A| < 3|N| .$$

Each arc of A has a non-negative flow cost, $C(i,j)$.

The graph G forms the basis upon which the vehicle routing model performs its function of routing vehicles from one location (node i) to another (node j) at minimum cost. With no restrictions placed upon i or j, all nodes in N are potential origins and destinations for the model. However, it is recognized that only a very small fraction of all the shortest paths in G will ever be used. So, rather than wasting considerable time to produce an all-pairs shortest path solution which cannot be stored with available computing machinery anyway, it is desired that the shortest path algorithm produce a part of the all-pairs solution (i.e., one small enough to be stored internally) in which some shortest paths are known and from which all others can be quickly approximated. With current technology, it is assumed that several $|N|$-length arrays can be stored internally but not $|N|$ of them.

To this end, a set of nodes is designated as a reference set. This set, R ($r=|R|$), is a subset of N and in practice it is expected that:

$$r << |N| \text{ (e.g., } r = .0001|N|).$$

This designation is a result of a partitioning of all the nodes of N into r clusters where each cluster contains one reference node. All of the remaining nodes in a cluster are considered ordinary nodes.

This designation of R is to be used by the shortest path algorithm to produce a subset of the all-pairs solution, namely the all-pairs of R solution. From this all-pairs of R solution, the vehicle routing model must be able to determine the shortest path between any nodes in the network. Thus, the all-pairs of R solution must be structured such that all nodes in the network are a known distance from at least one of the reference nodes. To facilitate this, the shortest path algorithm shall utilize an engineering parameter approach which provides the user a degree of control over the amount of approximation used.

The engineering parameter approach is defined as follows. Letting SP_{ij} represent the optimal shortest path cost from reference root node i to node j, and EP1, EP2, and EP3 the engineering parameters, with EP1 < EP2 < EP3, the proposed rules are as follows:

- if $SP_{ij} \leq EP1$, then the algorithm is required to produce the shortest path accurately
- if SP_{ij} is known and satisfies $EP1 < SP_{ij} < EP2$, then the algorithm is allowed to approximate SP_{ji} by SP_{ij}
- if $SP_{ij} \geq EP2$, then the algorithm may neglect computing the shortest path, approximate SP_{ij} by EP3, and force the algorithm to a halt (i.e., stop computing shortest paths)

The first rule requires that all shortest paths of length EP1 or less be computed accurately. That is, the optimal shortest path must be located and the node labeled appropriately if the node is to be labeled at all. In essence, this provides the means by which the user can ensure that the shortest path from a reference node to each of the ordinary nodes in the same cluster is accurately computed. Applied to each cluster, this ensures that all nodes in N are a known distance from at least one of the reference nodes.

The second rule, essentially provides for assumed symmetry between reference root node i and node j. With respect to the vehicle routing problem, this rule is designed to allow the algorithm to ignore the asymmetry of a particular route on trips of specified length. For example, on trips between location i in city 1 and location j in city 2, the one-way on-ramp to an interstate can be ignored as its distance is insignificant to the total shortest path distance between i and j. Thus to save work and execution time, SP_{ji} is approximated by SP_{ij} . In contrast, this rule insures that symmetry is not assumed when the path is less than EP1. Consider the case where node i and node j are different locations in the same business district in a city. To ignore one-way streets in this setting may produce a gross inaccuracy in the computed SP_{ij} Thus, the selection of EP1 and EP2 enable the user to determine under what conditions symmetry may be assumed.

The third rule, defines the maximum distance from the reference root node(s) that the user wants examined. When the algorithm encounters the first shortest path length greater than EP2, the associated node is labeled with a dummy distance (EP3) and the algorithm stops. All nodes, to include the non-root reference nodes, that are outside this maximum range are not labelled. Leaving these nodes unlabeled is acceptable since in vehicle routing these nodes will never be visited consecutively and thus it is not required to know SP_{ij} for them.

These rules essentially allow the user to adjust the scope of the shortest path problem according to individual desires or needs, as well as providing for flexibility to take advantage of advances in computer hardware as improvements are introduced.

2. **SP_{ij} Approximations**

The shortest path solutions produced are for all-pairs of R. That is, each node in R is designated as the root node once, and this results in r single-source shortest path solutions. From these r solutions, any SP_{ij} for G can be computed in one step. The user designates the i and j of interest, and the model knows the reference node that each is associated with. Letting I represent the reference node associated with node i, and J represent the reference node associated with node j, then the computation of any SP_{ij} is as follows:

$$SP_{ij} = pU^{I}(i) + qU^{J}(j) + U^{I}(J)$$

where p and q are weights designated by the user for adjusting this approximation.

3. **The Problem**

The problem to be addressed is two-fold. The first task is to determine what type of shortest path algorithm is most appropriate to this situation and will function as the base for construction of the reference node aggregation algorithm. And second is to design and implement an algorithm which reflects the engineering parameter approach and produces an all-pairs of R shortest path solution from which all SP_{ij} can be approximated efficiently. The goal is to produce the all-pairs of R solution quickly, from which the vehicle routing model can compute in one step any SP_{ij} .

C. THE PROPOSED ALGORITHM

1. Base Algorithm Selection

A straightforward approach to solving this problem efficiently is to choose an algorithm which can produce shortest paths without necessarily examining every arc in A and each node in N. The ultimate algorithm would examine only those nodes and arcs involved in the shortest paths for R.

In this pursuit, it was decided to use a label setting algorithm as the base upon which to build. The most attractive aspect of a label setting method is that at each iteration, the permanent labels are optimal. That is, the shortest paths computed from the specified root are part of the final shortest path tree $T=(N_T,A_T)$, even though T is not complete until the $|N|$-1 iteration. Label correcting, on the other hand, is not necessarily optimal until its last iteration. As well, an all-pairs algorithm (like Floyd's) is not optimal until all paths in the network have been examined. By exploiting this optimality feature of label setting, it is hoped that an efficient, yet effective, algorithm can be developed.

The reader will recall that in Chapter II, two label setting algorithms were discussed. The first was the basic label setting algorithm, while the second, improved label setting, used temporary labels which enabled a one-time examination of each arc vice repetitive examinations. Both of these label setting techniques will be used as a base for the reference node aggregation algorithm design, and testing will provide for a comparison between them.

2. Termination Measures

A means of exploiting the label setting algorithm for the problem at hand involves constructing the capability to force the termination of the algorithm before normal completion at the $|N|$-1'st iteration. The shortest path solution for a given reference root node is complete no later than the point where all non-root reference nodes are labeled, and thus when this occurs the algorithm can be stopped. This premature termination is acceptable due to the fact that the shortest paths are optimal at each iteration and that those shortest paths not identified by the time all reference nodes are labeled have no direct impact on the all-pairs of R shortest path solutions. At worst, an alternate shortest path may be ignored.

In essence, this involves adding one step to the base label setting algorithm which checks to see if the label of each reference node is permanently set. If they are, the algorithm is stopped. On the other hand, if there is even one reference node not labeled, the algorithm proceeds as normal.

The termination measure is designed to stop the algorithm from doing work that does not directly contribute to producing the desired shortest paths for the nodes in R. To assist in this effort, attention is now turned to another efficiency measure.

3. Avoiding Recomputation of Shortest Paths

Another measure to be added to the base algorithm also takes advantage of the nature of the label setting technique. As was mentioned earlier, solving a network for an all-pairs of R shortest path solution requires that the algorithm be run once for each reference node and that each reference node be designated as the root node for a specific run.

With the exception of the first reference root node, this algorithm repetition can be exploited in that some shortest paths do not have to be computed again. Each repetition of the algorithm locates and labels the non-root reference nodes along some shortest path. Should a non-root reference node have any successors in a particular solution, these successors can be immediately labeled at the beginning of the iteration where the node is designated as the reference root node. This occurs, once again due to the fact that the label setting technique produces shortest paths at each iteration. Thus, any successor (node) to node i on a shortest path where i is not the root node, is also a successor on that same path when node i is the root node.

So at each repetition of the algorithm, the previous shortest path solutions can be examined to determine if the new reference root node had successors in those solutions. When this occurs, the successors can be immediately labeled, thereby eliminating the computation of those shortest paths for the current iteration.

4. Summary

Two versions of the reference node aggregation algorithm are proposed. The first version utilizes the basic label setting algorithm (previously depicted in Figure 2.5 of Chapter II) as an underlying structure and blends in the engineering parameter approach, as well as the termination measures and the measures for avoiding the recomputation of shortest paths.

Likewise, the second version blends in the parameter approach and these same measures, but differs in that it utilizes the improved label setting algorithm (previously depicted in Figure 2.6 of Chapter II) as the underlying structure.

D. THE DESIGNED ALGORITHM

1. Data Structure

Supporting both versions of the proposed reference node aggregation algorithm is a data structure which is similar to that discussed in Chapter II. The network is represented internally by the TAIL array, the flowcost array C( ), and the head array H( ) (i.e., TO-NODE). As for the shortest path solutions, they are stored in the label and predecessor functions discussed previously. However, both functions are now defined as matrices of dimension $|R|$ by $|N|$ with each row containing the shortest path solution associated with the reference node used as the root node to generate that solution.

To round out this data structure, three additional arrays are introduced.

a. Reference Node Array

To identify which nodes in the network are designated as reference nodes, an array of length $|R|$ is defined. This array, RF( ), simply contains as its elements the node number of each node belonging to R, that is, those nodes who have been designated reference nodes.

b. Traversal and Depth Functions

The final two arrays added to the data structure are of length $|N| + 1$ and enable the identification of successors to non-root reference nodes in previous solutions. These well-known arrays are the traversal and depth functions [Ref. 2: page 15].

The traversal function IT( ) provides a means to keep track of the dynastic ordering of nodes in the shortest path tree T. This dynastic ordering produces a ring of nodes starting at the root with each entry in IT( ) pointing to the next node in succession until the final value points back to the root.

The depth function DP( ) keeps track of how many levels below the root node a non-root node is found in T. The root node is assigned a depth of zero. Those nodes directly attached to the root are assigned a depth of one and this level increases the further a node gets from the root. In essence, the depth of a node indicates the number of nodes (including the root) visited prior to reaching that non-root node along the shortest path.

For the reference node aggregation algorithm, depth and traversal are used in conjunction with each other to identify all the successors of a node. IT(i), the traversal value for node i, points to the next node in the dynastic ordering and the depth of that next node specifies whether the node is a successor to node i or merely of lower order as compared to node i. Iterating this, all the successors of node i can be identified as well as the shortest paths the successors are found on. These paths, thus identified, can be used in the current solution without the necessity of computing from scratch.

2. **The Implemented Versions of the Proposed Algorithm**

Computer implementation of the proposed algorithm was accomplished in FORTRAN. As was indicated earlier, both the basic and improved label setting techniques were used as base algorithms.

Figure 3.1 depicts the reference node aggregation algorithm utilizing the improved label setting as its base. In this algorithm, a shortest path solution for each reference node is generated. Thus, step 1 initializes the solution index and the associated reference root node is chosen in step 2. The predecessor, label, depth, and traversal functions are initialized for the current solution in step 3. Step 4 has two parts, and in part 4a, any SP_{jr} found in previous solutions (i.e., associated with another reference node) that has a length between EP1 and EP2, is used to approximate S_{rj} where r is the current reference root node. Step 4b sets the label and predecessor of all nodes which were successors of the current reference root node in the previous solutions. In step 5, the reference nodes are examined at each opportunity where it is possible that each has been permanently labeled, and upon finding this to be true, forces the current iteration to halt and a new iteration (with a new reference root node) to start. The forward star of the last labeled node is examined in step 6 and all temporary labels which can be improved upon are updated. The best temporary label is located in step 7, while step 8 determines if the shortest path solution has reached the furthest distance (EP2) from the root node that the user wants examined. If this distance is met or exceeded, the current iteration is halted, and a new reference root node is designated. In step 9, the best temporary label is set permanent. Step 10 increments the iteration index in preparation for selecting the new reference root node. And finally, in step 11 the all-pairs of R solution is complete and the algorithm stops.

Figure 3.2 depicts the reference node aggregation algorithm using basic label setting as its base. In this version, a shortest path solution for each reference node is

```
Input:
    (1) RF( ), the array of reference nodes.
    (2) Network data in the form of the arrays C( ), H( ), and T( ).
    (3) EP1, EP2, EP3
Output:
    (1) $P^I()$ for $I = 1, |R|$
    (2) $U^I()$ for $I = 1, |R|$
1. Initialize the repetition index: $I = 1$
2. Initialize the tree T with the reference root node: $r = RF(I)$
3. Iteration initialization:
    $P^I(j) = 0$, for all $j \in N$
    $U^I(j) = \infty$, for all $j \in N$
    $U^I(r) = 0$
    $DP^I(j) = 0$, for all $j \in N$
    $IT^I(j) = 0$, for all $j \in N$
    $DP^I(|N|+1) = -1$
    $IT^I(|N|+1) = r$
    $IT^I(r) = |N|+1$
    $RFCNT = |R|$
4. FOR $K = 1, I-1$ DO
    a. Examine the backpath of r.  Let $j = P^K(r)$
        WHILE $j \neq 0$ DO
            IF $EP1 < U^K(r) - U^K(j) < EP2$ THEN
                $U^I(j) = U^K(r) - U^K(j)$
                $P^I(j) = K + |N|$  (denotes that the path from r to j is found
                                     in the shortest path solution for RF(K))
                $j = P^K(j)$
            ENDIF
        END WHILE
```

Figure 3.1 Reference Node Aggregation Algorithm Using Improved Label Setting.

```
    b. Examine the successors of r.  Let $s = IT^K(r)$.

        WHILE $DP^K(r) < DP^K(s)$ . DO:
            $U^I(j) = U^K(r) - U^K(j)$
            $P^I(j) = P^K(j)$
            $s = IT^K(s)$.
        END WHILE
END DO
5. IF $P^I(j) > 0$ for all $j \in R$, $j \neq RF(I)$ THEN go directly to step 10
    ELSE set RFCNT = number of unlabeled reference nodes
6. Examine the forward star of node r.  For each arc from r to a node j
where, $U^I(r) + C(r,j) < U^I(j)$, set:
        $U^I(j) = U^I(r) + C(r,j)$
        $P^I(j) = -r$
7. DSMALL = $\infty$
    DO j = 1,|N|
            IF $P^I(j) < 0$ and $U^I(j) <$ DSMALL THEN
                DSMALL = $U^I(j)$
                k=j
            ENDIF
    END DO
8. IF $U^I(k) >$ EP2 THEN for each $j \in N$ where $P^I(j) < 0$ set:
        $P^I(j) = -P^I(j)$
        $U^I(j) =$ EP3
        go directly to step 10
    ENDIF
9. Set $P^I(k) = -P^I(k)$
    RFCNT = RFCNT - 1
    IF RFCNT = 0 THEN goto step 5
    ELSE goto step 6
```

Figure 3.1 Reference Node Aggregation Algorithm
Using Improved Label Setting (cont'd.).

```
10. I = I + 1
    IF I < |R| THEN repeat step 2
11. Quit.
```

Figure 3.1 Reference Node Aggregation Algorithm Using Improved Label Setting (cont'd.).

also generated. Steps 1 through 5 are identical to those for the reference aggregation algorithm using improved label setting. In step 6, the forward star of all labeled nodes is examined identifying those unlabeled nodes that are candidates to be labeled, while step 7 locates and labels the best candidate. Step 8 determines if the shortest path solution has reached the furthest distance (EP2) from the root node that the user wants examined. If this distance is met or exceeded, the current iteration is halted, and a new reference root node is designated. Step 9 determines if it is time to check to see if all the reference nodes have been labeled. The iteration index is incremented in step 10 in preparation for selecting the new reference root node. And finally, in step 11 the all-pairs of R solution is complete and the algorithm stops.

E. EXPERIMENTAL DESIGN AND TEST RESULTS

1. Purpose

Testing was designed to determine if the implemented algorithm worked properly and to enable comparison of the reference aggregation algorithm and the base label setting algorithms. Additionally, a brief examination of the data structure was conducted to determine if the data structure size met the stated problem restrictions.

2. Design

The algorithms were run on a set of sample problems generating execution times. A network of 1000 nodes was selected as the test case size with the number of arcs bound as stated in the assumptions. The test was designed as follows:

- NETGEN was used to generate four separate networks of the following dimensions:

  Network A: 1000 nodes and 1750 arcs
  Network B: 1000 nodes and 1750 arcs
  Network C: 1000 nodes and 3000 arcs
  Network D: 1000 nodes and 3000 arcs

- The LRND function of FORTRAN was used to generate four sets of four reference nodes.

```
Input:
    (1) RF( ), the array of reference nodes.
    (2) Network data in the form of the arrays C( ), H( ), and T( ).
    (3) EP1, EP2, EP3
Output:
    (1) $P^I()$ for $I = 1, |R|$
    (2) $U^I()$ for $I = 1, |R|$
1. Initialize the repetition index: $I = 1$
2. Initialize the tree T with the reference root node: $r = RF(I)$
3. Iteration initialization:
    $P^I(j) = 0$, for all $j \in N$
    $U^I(j) = \infty$, for all $j \in N$
    $U^I(r) = 0$
    $DP^I(j) = 0$, for all $j \in N$
    $IT^I(j) = 0$, for all $j \in N$
    $DP^I(|N|+1) = -1$
    $IT^I(|N|+1) = r$
    $IT^I(r) = |N|+1$
    $RFCNT = |R|$
4. FOR $K = 1, I-1$ DO
    a. Examine the backpath of r. Let $j = P^K(r)$
        WHILE $j \neq 0$ DO
            IF $EP1 < U^K(r) - U^K(j) < EP2$ THEN
                $U^I(j) = U^K(r) - U^K(j)$
                $P^I(j) = K+|N|$ (denotes that the path from r to j is found
                                  in the shortest path solution for RF(K))
                $j = P^K(j)$
            ENDIF
        END WHILE
```

Figure 3.2 Reference Node Aggregation Algorithm Using Basic Label Setting.

```
        b. Examine the successors of r.  Let $s = IT^K(r)$.
           WHILE $DP^K(r) < DP^K(j)$, DO:
              $U^I(j) = U^K(r) - U^K(j)$
              $P^I(j) = P^K(j)$
              $s = IT^K(s)$
           END WHILE
     END DO
     5. IF $P^I(j) > 0$ for all $j \in R$, $j \neq RF(I)$
        THEN go directly to step 10
        ELSE set RFCNT = number of unlabeled reference nodes
     6. Examine the forward star of all the labeled nodes and define:
        $S = \{(i,j): i \in N_T; j \in N-N_T, (i,j) \in A\}$
        IF $S = \{\ \}$ THEN proceed directly to step 9.
     7. Examine each element of S and:
        a. Find $(k,l) = \text{argmin}\ \{U^I(i) + C(i,j): (i,j) \in S\}$
        b. Set:
           $P^I(l) = k$
           $U^I(l) = U^I(k) + C(k,l)$
     8.  IF $U^I(l) > EP2$ THEN
           $U^I(l) = EP3$
           go directly to step 10
        ENDIF
     9. RFCNT = RFCNT - 1
        IF RFCNT = 0  THEN goto step 5
        ELSE goto step 6
        ENDIF
     10. $I = I + 1$
           IF $I < |R|$  THEN repeat step 2
     11.  Quit.
```

Figure 3.2 Reference Node Aggregation Algorithm Using Basic Label Setting (cont'd.).

- Each algorithm was run against the four networks for each of the four sets of reference nodes. Table 3 identifies the test format.
- The time each algorithm took to compute the shortest path solutions for each set of reference root nodes was recorded.
- The execution results provide a complete block design which was be used in the non-parametric Friedman Test [Ref. 9: page 299] which examines the hypothesis that mean execution times for the various algorithms are identical.

TABLE 3

TEST FORMAT

	Sample Network				Eng. Parameters			
	A	B	C	D	EP1	EP2	EP3	
Test 1	1,2	1,2	1,2	1,2	N/A	N/A	N/A	
Test 2	3,4	3,4	3,4	3,4	9991	9992	9999	(Exact Solution Required)
Test 3	3,4	3,4	3,4	3,4	50	60	100	
Test 4	3,4	3,4	3,4	3,4	25	30	50	

Algorithm used:

1	Basic Label Setting Algorithm
2	Improved Label Setting Algorithm
3	Reference Aggregation Algorithm using Basic Label Setting
4	Reference Aggregation Algorithm using Improved Label Setting

3. Data Structure Size

The implemented design for the sample problems meets the memory assumptions of the problem statement. Table 4 provides a summary of the data structure size for the reference node aggregation algorithm. It should be noted that the implemented reference node algorithm retains previous shortest path solutions in main memory (for speed).

TABLE 4

DATA STRUCTURE REQUIREMENTS

Function	Label Setting Base	Reference Aggregation
Storing Network Data	$2\|A\| + \|N\|$	$2\|A\| + \|N\|$
Shortest Path Generation	---	$\|R\| + 2\|N\|$
Storing Shortest Path Solutions	$2\|N\|$	$2\|R\|\|N\|$
Total	$2\|A\| + 3\|N\|$	$2\|A\| + 3\|N\| + 2\|R\|\|N\| + \|R\|$

4. Algorithm Execution Time Comparison

Four tests were conducted to enable a comparison of algorithm execution times for the sample networks. Test 1 was designed to provide sample execution times for the base algorithms, namely basic and improved label setting. Tests 2, 3, and 4 were designed to provide sample execution times for both versions of the reference node algorithm using differing values of the engineering parameters. Test 2 uses engineering parameters that for the particular networks involved can be considered as infinite values since no path approached the specified length. Thus, test 2 essentially examines the reference node aggregation algorithm where the engineering parameters have no impact on the shortest path solutions. Tests 3 and 4 use engineering parameter values that restrict shortest path solutions subject to the designed rules.

The data produced by Tests 1 through 4 consisted of the accumulated time it took each algorithm to solve the shortest path for a given set of four reference nodes. Thus, each test produced 32 data points.

The Friedman Test [Ref. 9: page 299] is a non-parametric test which makes no distributional assumptions. It utilizes a randomized complete block design to test the null hypothesis that treatment effects are equal, with the alternate hypothesis that at least two effects are not equal. To this end, the algorithms were considered treatments, while the reference set/sample network pairs were blocks. Thus, the randomized complete block design consists of eight treatments and sixteen blocks. Figure 3.3 presents the data in the randomized complete block format utilized for the Friedman Test. In this case, the null hypothesis translates to that the mean time of execution to produce a shortest path solution for a given network and reference node set is the same regardless of the algorithm used. The alternate then becomes that at least two of the algorithm implementations have different mean execution times.

Utilizing an α-level of 0.05, the Friedman test statistic was computed giving T2 = 113.6. An F-statistic with (7,105) degrees of freedom approximates T2. With F(7,105) for $\alpha = 0.05$ equal to 2.109, the null hypothesis was rejected enabling use of the multiple comparison extension of the Friedman Test [Ref. 9: page 297]. This multiple comparison showed, for the sample networks, that none of the treatment effects were equal statistically. Thus, the implemented algorithm is a robust one.

Further, this comparison indicated that for the sample problems, the reference node aggregation algorithm utilizing the improved label setting base outperformed (i.e., was faster) that version which used the basic label setting as a base. Table 5 provides a summary of the algorithm performance for the sample networks.

5. Conclusions

The design and implementation of the reference node aggregation algorithm has been successfully accomplished. In addition, the testing showed that:

- Both label setting and improved label setting served as an adequate base for the algorithm implementation, and the resulting reference node aggregation algorithm functioned as planned.
- The performance of the improved label setting base was superior to that of the basic label setting as implemented in the reference node aggregation algorithm.
- The engineering parameter approach is a robust one and demonstrated its ability to enable the user to adjust the reference node aggregation algorithm with respect to the scope of the shortest path solutions produced.

Hypothesis:

H_0 : The treatments have identical effects within a block.

H_1 : At least one treatment tends to yield larger observed values than another treatment.

Note: Each data entry is the execution time in CPU seconds for an algorithm (i.e., treatment) to solve the all-pairs of R shortest path problem with |R| = 4 reference nodes, given networks of dimensions specified in the block definition below.

	Treatment							
	1	2	3	4	5	6	7	8
Block								
1	27.17	11.13	15.31	10.87	9.12	5.12	8.51	0.44
2	27.43	10.97	13.15	10.67	12.24	5.93	5.93	0.39
3	27.74	11.02	20.20	10.70	15.43	3.55	15.10	0.21
4	27.70	11.01	13.76	10.84	17.61	3.28	17.79	0.25
5	27.91	11.14	18.49	11.20	17.96	5.57	20.00	0.56
6	26.70	10.98	22.12	11.35	11.51	3.91	17.99	0.26
7	26.52	11.09	22.23	10.98	14.95	5.28	17.00	0.51
8	26.73	10.83	11.31	11.08	11.36	6.07	5.41	0.38
9	38.59	12.12	28.58	11.65	28.83	11.75	21.47	6.36
10	38.47	11.96	30.92	11.64	31.11	11.70	23.42	8.89
11	38.86	11.86	27.18	11.62	27.70	11.87	18.78	9.42
12	37.89	11.81	23.21	11.54	23.16	11.67	19.69	7.71
13	37.81	11.97	19.29	11.90	18.67	11.85	20.95	6.47
14	38.10	11.91	15.12	11.85	14.58	11.76	14.46	8.81
15	38.29	11.98	15.07	11.78	14.28	11.55	14.69	9.14
16	38.62	11.83	28.03	11.79	27.04	11.46	18.70	5.32

Treatment Definitions:

1	Basic Label Setting Algorithm
2	Improved Label Setting Algorithm
3	Reference Node Aggregation Algorithm using Basic Label Setting with EP1=9991, EP2=9992, and EP3=9999.
4	Reference Node Aggregation Algorithm using Improved Label Setting with EP1=9991, EP2=9992, and EP3=9999.
5	Reference Node Aggregation Algorithm using Basic Label Setting with EP1=50, EP2=60, and EP3=100.
6	Reference Node Aggregation Algorithm using Improved Label Setting with EP1=50, EP2=60, EP3=100.

Figure 3.3 Randomized Complete Block Design for the Friedman Test.

Treatment Definition (con't):

7	Reference Node Aggregation Algorithm using Basic Label Setting with EP1=25, EP2=30, and EP3=50.
8	Reference Node Aggregation Algorithm using Improved Label Setting with EP1=25, EP2=30, and EP3=50.

Block Definitions:

1	Network A: 1000 nodes, 1750 arcs, Reference Set 1
2	Network A: 1000 nodes, 1750 arcs, Reference Set 2
3	Network A: 1000 nodes, 1750 arcs, Reference Set 3
4	Network A: 1000 nodes, 1750 arcs, Reference Set 4
5	Network B: 1000 nodes, 1750 arcs, Reference Set 1
6	Network B: 1000 nodes, 1750 arcs, Reference Set 2
7	Network B: 1000 nodes, 1750 arcs, Reference Set 3
8	Network B: 1000 nodes, 1750 arcs, Reference Set 4
9	Network C: 1000 nodes, 3000 arcs, Reference Set 1
10	Network C: 1000 nodes, 3000 arcs, Reference Set 2
11	Network C: 1000 nodes, 3000 arcs, Reference Set 3
12	Network C: 1000 nodes, 3000 arcs, Reference Set 4
13	Network D: 1000 nodes, 3000 arcs, Reference Set 1
14	Network D: 1000 nodes, 3000 arcs, Reference Set 2
15	Network D: 1000 nodes, 3000 arcs, Reference Set 3
16	Network D: 1000 nodes, 3000 arcs, Reference Set 4

Figure 3.3 Randomized Complete Block Design for the Friedman Test. (cont'd.)

TABLE 5

SUMMARY OF ALGORITHM PERFORMANCE
(CPU SECS REFERENCE SET)

	Network A & B		Network C & D	
	Sample Mean	Sample Standard Deviation	Sample Mean	Sample Standard Deviation
Treatment				
1	27.24	0.54	38.33	0.37
2	11.02	0.24	11.93	0.25
3	17.07	4.31	23.43	6.25
4	10.96	0.24	11.72	0.13
5	14.15	3.22	23.17	6.59
6	4.84	1.95	11.70	0.14
7	9.72	7.12	19.02	3.14
8	0.38	0.13	7.69	1.54

Treatment:

1	Basic Label Setting Algorithm
2	Improved Label Setting Algorithm
3	Reference Node Aggregation Algorithm using Basic Label Setting with EP1=9991, EP2=9992, and EP3=9999.
4	Reference Node Aggregation Algorithm using Improved Label Setting with EP1=9991, EP2=9992, and EP3=9999.
5	Reference Node Aggregation Algorithm using Basic Label Setting with EP1=50, EP2=60, and EP3=100.
6	Reference Node Aggregation Algorithm using Improved Label Setting with EP1=50, EP2=60, EP3=100.
7	Reference Node Aggregation Algorithm using Basic Label Setting with EP1=25, EP2=30, and EP3=50.
8	Reference Node Aggregation Algorithm using Improved Label Setting with EP1=25, EP2=30, and EP3=50.

IV. CONCLUSIONS

A. SOTACA

Chapter II presented two problems that have arisen in the use of the OJCS contingency planning model SOTACA and proposed some means of resolving them. SOTACA uses an implementation of Floyd's algorithm to compute an all-pairs shortest path solution and experiences slow execution (i.e., upwards of twenty minutes on a MICRO VAX) when dealing with networks at or near the maximum allowable size which is very small by contemporary standards. The author has proposed that this first problem be resolved by modifying SOTACA so that a single-source shortest path algorithm, label setting or label correcting, is used to produce shortest path solutions on demand vice the current method preprocessing all pairs. Accompanying this algorithm change, it has been proposed that the SOTACA network representation be changed to a forward star format because of the gained efficiencies. Testing showed that these algorithms and the forward star network representation produced shortest path solutions very quickly even for networks at the maximum allowable size. With these changes, it is anticipated that SOTACA's slow execution problem will be resolved. However, the reader should be aware that other implementations of the single-source shortest path algorithms exist in the literature available, and could also be applied to this time problem.

As for SOTACA's second problem, that of nonrecognition of alternate shortest paths, the use of a depth-first search (and the modified label setting algorithm) has been shown to correctly locate and enumerate alternate shortest paths. However, there are several research and implementation issues which this thesis has only touched upon. Some of these issues that the author feels should be examined are:

(1) the impact on SOTACA of the additional code and data structure required to implement a means for enumerating alternate shortest paths,

(2) the effects of this added capability with respect to model execution (i.e., time), and

(3) a comparison (speed, data structure size, source code size, . . .) of the depth-first search versus the breadth-first search, or any other methodologies for enumerating alternate shortest paths.

Nonetheless, what has been shown is that the alternate shortest path problem can be resolved and that methods to address it are readily available.

B. REFERENCE NODE AGGREGATION

In Chapter III, a new algorithm, reference node aggregation, was proposed. This algorithm is designed to produce a subset of the all-pairs shortest path solution for large scale networks. This subset solution, an all-pairs of R solution, is specifically structured and is intended to support a vehicle routing model by providing the means by which it can quickly compute (i.e., in one step) the approximate cost of the shortest path between any two nodes in the network. Additionally, the algorithm provides for user-specification of three engineering parameters. These parameters can be used to make tradeoffs between the accuracy of the all-pairs of R solution and the total time it takes to produce it.

The algorithm was implemented in two forms, one using a basic label setting methodology, and the other using an improved (one look per arc) label setting methodology. Testing demonstrated that both implementations were successful and that the improved label setting methodology was superior as its production of the subset solution was significantly faster. Also, the engineering parameter approach proves to be flexible and enables the user to adjust the the algorithm to fit the individual problem as well as providing a means to take advantage of computer hardware improvements without modifying the algorithm.

However, the author feels that the reference node aggregation algorithm, as presented in this thesis, can be improved upon. Specifically, the following are potential areas of improvement:

(1) Determining which and how many of the previous solutions to examine when looking for successors to a reference root node. The current implementation examines them sequentially.

(2) Determining which and how many of the previous solutions to examine when approximating a SP_{ij} by SP_{ji}

(3) Reorganizing the sequence of the steps so that the algorithm is provided some "room to run" before it starts expending effort to check on whether all reference nodes are labeled, or whether the labeling has reached the maximum distance from the root to be examined.

The main point is that this thesis has concentrated on showing that the reference node aggregation concept (with its engineering parameter approach) works, and that additional analysis could make improvements to the functioning of the algorithm.

Beyond improvements, the next step (though not part of this thesis) is the major one which involves embedding the reference node aggregation algorithm in a vehicle routing model and then assessing its performance. In this way, the validity of the reference node aggregation algorithm with its engineering parameter approach can be shown.

LIST OF REFERENCES

1. *State Of The Art Contingency Analysis (SOTACA) Analyst's Guide to Theory (Preliminary Draft)*, Joint Analysis Directorate, Organization of the Joint Chiefs of Staff, Washington D.C., March 1986.

2. Bradley, Gordon H., Brown G.G., and Graves G.W., "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, Volume 25, Number 1, September 1977.

3. Dial, R., Glover, F., Karney, D., and Klingman, D., "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," *Networks*, Volume 9, 1979.

4. Aho, Alfred V., Hopcroft J.E., and Ullman J.D., *Data Structures and Algorithms*, Addison-Wesley Publishing Company, 1985.

5. Author's Personal Class Notes from Lectures by Gerald G. Brown, OA4202, Network Flows and Graphs, Naval Postgraduate School, Monterey, California, September-December 1986.

6. Glover, F., Klingman, D., and Phillips, N., "A New Polynomially Bounded Shortest Path Algorithm," *Operations Research*, Volume 33, Number 1, January-February 1985.

7. Winston, Patrick Henry, *Artificial Intelligence, Second Edition*, Addison-Wesley Publishing Company, 1984.

8. Klingman, D., Napier, A. and Stutz, J., "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," *Management Science*, Volume 20, Number 5, January 1974.

9. Conover, W. J., *Practical Nonparametric Statistics, Second Edition*, John Wiley and Sons, 1980.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	J-8, Organization of the Joint Chiefs of Staff Room R1D940 The Pentagon Arlington, Virginia 20301-5000	1
4.	Professor Richard E. Rosenthal Code 55Rl Naval Postgraduate School Monterey, California 93943-5004	1
5.	Captain Jerome W. Brown, Jr. 13814 Leighfield Street Chantilly, Virginia 22021	1